“My name is Ryan; I *inherited* the ship from the previous Dread Pirate Roberts, just as you will *inherit* it from me. The man I *inherited* it from is not the real Dread Pirate Roberts either. His name was Cummerbund. The real Roberts has been retired 15 years and living like a king in Patagonia”.

INHERITANCE

Boaz Kantor

Introduction to Computer Science

IDC Herzliya (“Reichman”)

INTERFACES VS. INHERITANCE

- Interfaces:
  - Set the interaction between an object and its clients.
  - Allow conformity and standardization in a project.
  - Thumb rule: everything that is "able" should be an interface.
  - Thumb rule: an abstract class with nothing but abstract methods should be an interface.
- Inheritance:
  - Setting hierarchy of specific-general classes.
- A class can inherit up to one class, and implement multiple interfaces.

INTERFACES

INTERFACES, GROUND RULES

- An interface defines an object API.
- All interface methods are **`public abstract`**.
- Any class implementing an interface must implement all of its methods.
- Here's the beauty part:
  - A client can use the interface as a type, ignoring the type of the class which implemented it (polymorphism).

INTERFACES, EXAMPLE

```
public interface SomeInterface {
        void foo();
        String youMustImplementMe(float x);
}
```

INTERFACES, EXAMPLE

```
public class SomeClass implements SomeInterface {
	public void foo() {
		System.out.println("SomeClass.foo()");
	}
	public String youMustImplementMe(float y) {
		System.out.println("SomeClass.youMustImplementMe()");
		return null;
	}
	public void unrelatedMethod() {
		System.out.println("SomeClass.unrelatedMethod()");
	}
}
```

INTERFACES, EXAMPLE

```java
public class SomeOtherClass {
	public static void main(String[] args) {
		SomeClass s = new SomeClass();
		f(s);
	}
	private static void f(SomeInterface i) {
		i.foo();
	}
}
```

INHERITANCE

INHERITANCE, GROUND RULES

1. The derived class includes all members of the super class.
2. Non-private members of super class can be accessed directly by sub classes.
3. The derived class can extend the functionality of the super class.
4. The derived class can override the functionality of the super class.
5. Overridden method implementation has precedence over super class implementation.
6. Overriding fields and static methods is called "hiding" and is discouraged.
7. Constructors are not derived.

INHERITANCE, GROUND RULES

8. Any constructor must first call a super constructor, directly or by calling another constructor.
9. If no such call is written, **`super()`** is implicitly called.
10. Anything "protected" is visible to other classes of the same package and derived classes, no matter to which package they belong.
11. You can increase a super method visibility, but can't reduce it.
12. A class which does not inherit another class, automatically inherits **`java.lang.Object`**.
13. **`final`** methods, classes and fields cannot be overridden.
14. There is no multiple inheritance in Java.

INHERITANCE BEST PRACTICES

1. Avoid writing protected instance variables.
    1. Instance variables should always be private.
    2. Use protected/public setters and getters.
2. Any method called directly or indirectly by a constructor should be final.
3. Always use **`instanceof`** before casting a reference (polymorphism).

ABSTRACT

THE LOGICS BEHIND

- Assume a general class G and subclasses A and B.
- Sometimes there is no meaning to hold an object of type G.
- G should still implement default behavior in methods.
- In addition, G should define 'an interface' for subclasses.
- A and B must implement their own implementation of the 'interface' (i.e. abstract) methods.
- A and B still inherit the default behavior of the implemented methods.

GROUND RULES

1. An abstract class cannot be instantiated.
2. An abstract method cannot be implemented.
3. A class with an abstract method must be defined abstract.
4. A sub-class which does not implement a super's abstract method is abstract.
5. An abstract class with no implementation should probably be an interface.

ABSTRACT, EXAMPLE

- Assume class **`MotorizedVehicle`**, with subclasses **`Motorcycle`** and **`Airplane`**.
- There is no meaning to an object of type **`MotorizedVehicle`**.
- However, since all motorized vehicles have an engine, **`MotorizedVehicle`** should have an engine implementation.
- Since all motorized vehicles need to start the engine, **`MotorizedVehicle`** should have an implemented method **`startEngine`**.
- Since each motorized vehicle has its own turning implementation, **`MotorizedVehicle`** should have an abstract method **`turn()`**.
- **`Motorcycle`** and **`Airplane`** should only implement **`turn()`**. They can override **`startEngine()`**, if they wish.

ABSTRACT, EXAMPLE

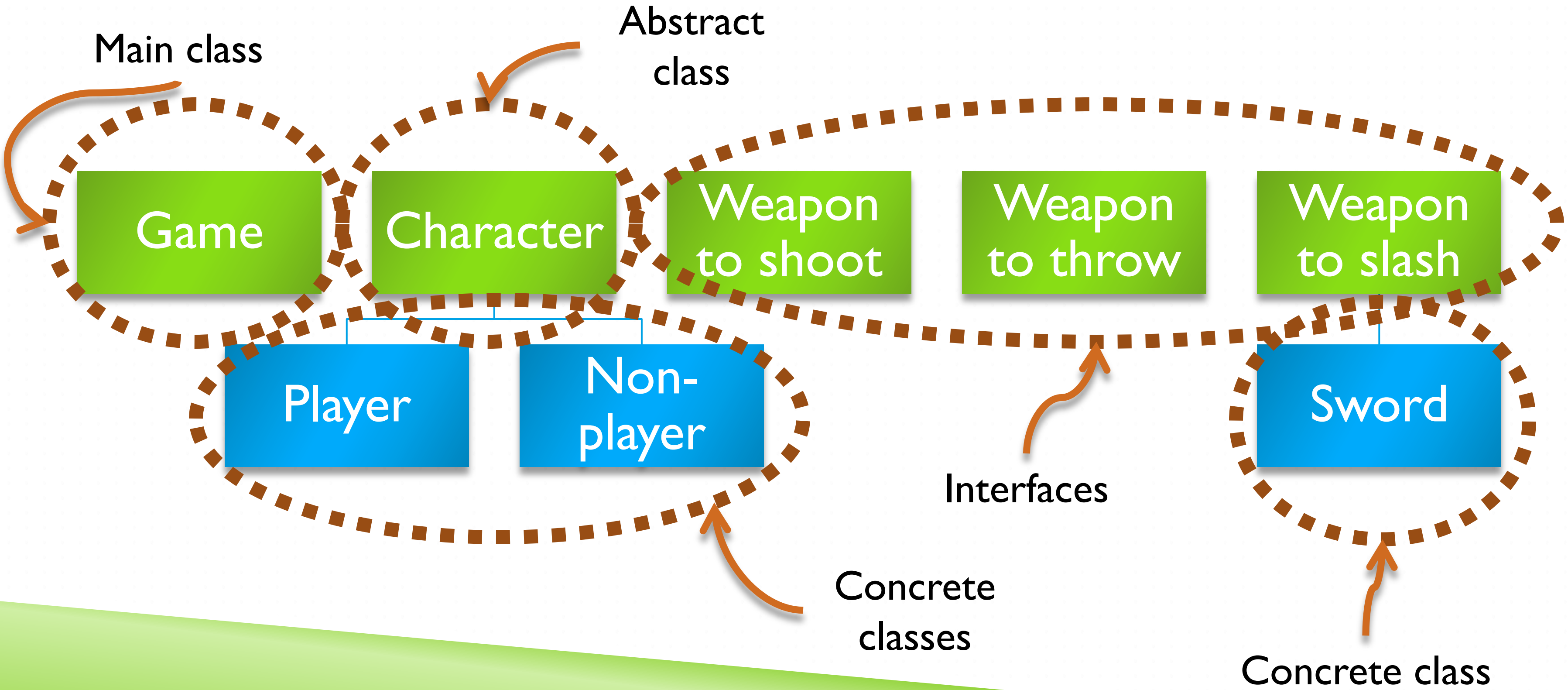
```
public abstract class MotorizedVehicle {
      private Engine engine = new Engine();
      public void startEngine() {
            engine.start();
      }
      public abstract void turn(Direction d);
}
```

INHERITANCE, EXAMPLE

PROJECT REQUIREMENTS

- Write an MMORPG (Massive Multiplayer Online Role Playing Game).
- The game consists of players and computer characters.
- Implement game, characters and weapons.

HIGH LEVEL DESIGN


Main class
Abstract class
Game
Character
Weapon to shoot
Weapon to throw
Weapon to slash
Player
Non-player
Sword
Interfaces
Concrete classes
Concrete class